UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

DTIC FILE COPY

(2)

AD-A205 658

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Ada Compiler Validation Summary Report: TELESOFT CRAY Ada Compiler, Release 1.0 CRAY-2 Host and Target		5. TYPE OF REPORT & PERIOD COVERED 12 June 1988 to 12 June 1988
		6. PERFORMING ORG. REPORT NUMBER -
7. AUTHOR(s) Wright-Patterson AFB, OH		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION AND ADDRESS Wright-Patterson AFB, OH		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Ada Joint Program Office United States Department of Defense Washington, DC 20301-3081		12. REPORT DATE
		13. NUMBER OF PAGES
14. MONITORING AGENCY NAME & ADDRESS (If different from Controlling Office) Wright-Patterson AFB, OH		15. SECURITY CLASS (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20 If different from Report)

UNCLASSIFIED

DTIC ELECTE MAR 0 2 1989 D

H

18. SUPPLEMENTARY NOTES

19. KEYWORDS (Continue on reverse side if necessary and identify by block number)

Ada Programming language, Ada Compiler Validation Summary Report, Ada Compiler Validation Capability, ACVC, Validation Testing, Ada Validation Office, AVO, Ada Validation Facility, AVF, ANSI/MIL-STD-1815A, Ada Joint Program Office, AJPO

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

CRAY Ada Compiler, Release 1.0, TELESOFT, Wright-Patterson AFB, CRAY-2 under UNICOS, Version 4.0 (Host) to CRAY-2 under UNICOS, Version 4.0 (Target), ACVC 1.9.

DD FORM 1473 EDITION OF 1 NOV 65 IS OBSOLETE
1 JAN 73 S/N 0102-LF-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Ada Compiler Validation Summary Report:

Compiler Name: CRAY Ada Compiler, Release 1.0

Certificate Number: 880610W1.09100

Host:
CRAY-2 under
UNICOS, Version 4.0

Target:
CRAY-2 under
UNICOS, Version 4.0

Testing Completed 12 June 1988 Using ACVC 1.9

This report has been reviewed and is approved.

Ada Validation Facility
Steven P. Wilson
Technical Director
ASD/SCEL
Wright-Patterson AFB OH 45433-6503

Ada Validation Organization
Dr. John F. Kramer
Institute for Defense Analyses
Alexandria VA 22311

Ada Joint Program Office
Virginia L. Castor
Director
Department of Defense
Washington DC 20301

AVF Control Number: AVF-VSR-191.1188
88-02-11-TEL

Ada COMPILER
VALIDATION SUMMARY REPORT:
Certificate Number: 880610W1.09100
TELESOFT
CRAY Ada Compiler, Release 1.0
CRAY-2

Completion of On-Site Testing:
12 June 1988

Prepared By:
Ada Validation Facility
ASD/SCEL
Wright-Patterson AFB OH 45433-6503

Prepared For:
Ada Joint Program Office
United States Department of Defense
Washington DC 20301-3081

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

This Validation Summary Report (VSR) describes the extent to which a specific Ada compiler conforms to the Ada Standard, ANSI/MIL-STD-1815A. This report explains all technical terms used within it and thoroughly reports the results of testing this compiler using the Ada Compiler Validation Capability (ACVC). An Ada compiler must be implemented according to the Ada Standard, and any implementation-dependent features must conform to the requirements of the Ada Standard. The Ada Standard must be implemented in its entirety, and nothing can be implemented that is not in the Standard.

Even though all validated Ada compilers conform to the Ada Standard, it must be understood that some differences do exist between implementations. The Ada Standard permits some implementation dependencies--for example, the maximum length of identifiers or the maximum values of integer types. Other differences between compilers result from the characteristics of particular operating systems, hardware, or implementation strategies. All the dependencies observed during the process of testing this compiler are given in this report.

The information in this report is derived from the test results produced during validation testing. The validation process includes submitting a suite of standardized tests, the ACVC, as inputs to an Ada compiler and evaluating the results. The purpose of validating is to ensure conformity of the compiler to the Ada Standard by testing that the compiler properly implements legal language constructs and that it identifies and rejects illegal language constructs. The testing also identifies behavior that is implementation dependent but permitted by the Ada Standard. Six classes of tests are used. These tests are designed to perform checks at compile time, at link time, and during execution.

1.1 PURPOSE OF THIS VALIDATION SUMMARY REPORT

This VSR documents the results of the validation testing performed on an Ada compiler. Testing was carried out for the following purposes:

- To attempt to identify any language constructs supported by the compiler that do not conform to the Ada Standard
- To attempt to identify any language constructs not supported by the compiler but required by the Ada Standard
- To determine that the implementation-dependent behavior is allowed by the Ada Standard

Testing of this compiler was conducted by SofTech, Inc. under the direction of the AVF according to procedures established by the Ada Joint Program Office and administered by the Ada Validation Organization (AVO). On-site testing was completed 12 June 1988 at Mendota Heights MN.

1.2 USE OF THIS VALIDATION SUMMARY REPORT

Consistent with the national laws of the originating country, the AVO may make full and free public disclosure of this report. In the United States, this is provided in accordance with the "Freedom of Information Act" (5 U.S.C. #552). The results of this validation apply only to the computers, operating systems, and compiler versions identified in this report.

The organizations represented on the signature page of this report do not represent or warrant that all statements set forth in this report are accurate and complete, or that the subject compiler has no nonconformities to the Ada Standard other than those presented. Copies of this report are available to the public from:

Ada Information Clearinghouse
Ada Joint Program Office
OUSDRE
The Pentagon, Rm 3D-139 (Fern Street)
Washington DC 20301-3081

or from:

Ada Validation Facility
ASD/SCEL
Wright-Patterson AFB OH 45433-6503

Questions regarding this report or the validation test results should be directed to the AVF listed above or to:

Ada Validation Organization
Institute for Defense Analyses
1801 North Beauregard Street
Alexandria VA 22311

1.3 REFERENCES

1. Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987.

2. Ada Compiler Validation Procedures and Guidelines, Ada Joint Program Office, 1 January 1987.

3. Ada Compiler Validation Capability Implementers' Guide, SofTech, Inc., December 1986.

4. Ada Compiler Validation Capability User's Guide, December 1986.

1.4 DEFINITION OF TERMS

ACVC — The Ada Compiler Validation Capability. The set of Ada programs that tests the conformity of an Ada compiler to the Ada programming language.

Ada Commentary — An Ada Commentary contains all information relevant to the point addressed by a comment on the Ada Standard. These comments are given a unique identification number having the form AI-ddddd.

Ada Standard — ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987.

Applicant — The agency requesting validation.

AVF — The Ada Validation Facility. The AVF is responsible for conducting compiler validations according to procedures contained in the Ada Compiler Validation Procedures and Guidelines.

AVO	The Ada Validation Organization. The AVO has oversight authority over all AVF practices for the purpose of maintaining a uniform process for validation of Ada compilers. The AVO provides administrative and technical support for Ada validations to ensure consistent practices.
Compiler	A processor for the Ada language. In the context of this report, a compiler is any language processor, including cross-compilers, translators, and interpreters.
Failed test	An ACVC test for which the compiler generates a result that demonstrates nonconformity to the Ada Standard.
Host	The computer on which the compiler resides.
Inapplicable test	An ACVC test that uses features of the language that a compiler is not required to support or may legitimately support in a way other than the one expected by the test.
Passed test	An ACVC test for which a compiler generates the expected result.
Target	The computer for which a compiler generates code.
Test	A program that checks a compiler's conformity regarding a particular feature or a combination of features to the Ada Standard. In the context of this report, the term is used to designate a single test, which may comprise one or more files.
Withdrawn test	An ACVC test found to be incorrect and not used to check conformity to the Ada Standard. A test may be incorrect because it has an invalid test objective, fails to meet its test objective, or contains illegal or erroneous use of the language.

1.5 ACVC TEST CLASSES

Conformity to the Ada Standard is measured using the ACVC. The ACVC contains both legal and illegal Ada programs structured into six test classes: A, B, C, D, E, and L. The first letter of a test name identifies the class to which it belongs. Class A, C, D, and E tests are executable, and special program units are used to report their results during execution. Class B tests are expected to produce compilation errors. Class L tests are expected to produce compilation or link errors.

Class A tests check that legal Ada programs can be successfully compiled and executed. There are no explicit program components in a Class A test to check semantics. For example, a Class A test checks that reserved words of another language (other than those already reserved in the Ada language) are not treated as reserved words by an Ada compiler. A Class A test is

passed if no errors are detected at compile time and the program executes to produce a PASSED message.

Class B tests check that a compiler detects illegal language usage. Class B tests are not executable. Each test in this class is compiled and the resulting compilation listing is examined to verify that every syntax or semantic error in the test is detected. A Class B test is passed if every illegal construct that it contains is detected by the compiler.

Class C tests check that legal Ada programs can be correctly compiled and executed. Each Class C test is self-checking and produces a PASSED, FAILED, or NOT APPLICABLE message indicating the result when it is executed.

Class D tests check the compilation and execution capacities of a compiler. Since there are no capacity requirements placed on a compiler by the Ada Standard for some parameters--for example, the number of identifiers permitted in a compilation or the number of units in a library--a compiler may refuse to compile a Class D test and still be a conforming compiler. Therefore, if a Class D test fails to compile because the capacity of the compiler is exceeded, the test is classified as inapplicable. If a Class D test compiles successfully, it is self-checking and produces a PASSED or FAILED message during execution.

Each Class E test is self-checking and produces a NOT APPLICABLE, PASSED, or FAILED message when it is compiled and executed. However, the Ada Standard permits an implementation to reject programs containing some features addressed by Class E tests during compilation. Therefore, a Class E test is passed by a compiler if it is compiled successfully and executes to produce a PASSED message, or if it is rejected by the compiler for an allowable reason.

Class L tests check that incomplete or illegal Ada programs involving multiple, separately compiled units are detected and not allowed to execute. Class L tests are compiled separately and execution is attempted. A Class L test passes if it is rejected at link time--that is, an attempt to execute the main program must generate an error message before any declarations in the main program or any units referenced by the main program are elaborated.

Two library units, the package REPORT and the procedure CHECK_FILE, support the self-checking features of the executable tests. The package REPORT provides the mechanism by which executable tests report PASSED, FAILED, or NOT APPLICABLE results. It also provides a set of identity functions used to defeat some compiler optimizations allowed by the Ada Standard that would circumvent a test objective. The procedure CHECK_FILE is used to check the contents of text files written by some of the Class C tests for chapter 14 of the Ada Standard. The operation of REPORT and CHECK_FILE is checked by a set of executable tests. These tests produce messages that are examined to verify that the units are operating correctly. If these units are not operating correctly, then the validation is not attempted.

The text of the tests in the ACVC follow conventions that are intended to ensure that the tests are reasonably portable without modification. For example, the tests make use of only the basic set of 55 characters, contain lines with a maximum length of 72 characters, use small numeric values, and place features that may not be supported by all implementations in separate tests. However, some tests contain values that require the test to be customized according to implementation-specific values--for example, an illegal file name. A list of the values used for this validation is provided in Appendix C.

A compiler must correctly process each of the tests in the suite and demonstrate conformity to the Ada Standard by either meeting the pass criteria given for the test or by showing that the test is inapplicable to the implementation. The applicability of a test to an implementation is considered each time the implementation is validated. A test that is inapplicable for one validation is not necessarily inapplicable for a subsequent validation. Any test that was determined to contain an illegal language construct or an erroneous language construct is withdrawn from the ACVC and, therefore, is not used in testing a compiler. The tests withdrawn at the time of this validation are given in Appendix D.

CHAPTER 2

CONFIGURATION INFORMATION

2.1 CONFIGURATION TESTED

The candidate compilation system for this validation was tested under the following configuration:

Compiler: CRAY Ada Compiler, Release 1.0

ACVC Version: 1.9

Certificate Number: 880610W1.09100

Host Computer:

Machine:	CRAY-2
Operating System:	UNICOS, Version 4.0
Memory Size:	128 Megawords

Target Computer:

Machine:	CRAY-2
Operating System:	UNICOS, Version 4.0
Memory Size:	128 Megawords

2.2 IMPLEMENTATION CHARACTERISTICS

One of the purposes of validating compilers is to determine the behavior of a compiler in those areas of the Ada Standard that permit implementations to differ. Class D and E tests specifically check for such implementation differences. However, tests in other classes also characterize an implementation. The tests demonstrate the following characteristics:

- Capacities.

  The compiler correctly processes tests containing loop statements nested to 65 levels, block statements nested to 65 levels, and recursive procedures separately compiled as subunits nested to 17 levels. It correctly processes a compilation containing 723 variables in the same declarative part. (See tests D55A03A..H (8 tests), D56001B, D64005E..G (3 tests), and D29002K.)

- Universal integer calculations.

  An implementation is allowed to reject universal integer calculations having values that exceed SYSTEM.MAX_INT. This implementation processes 64 bit integer calculations. (See tests D4A002A, D4A002B, D4A004A, and D4A004B.)

- Predefined types.

  This implementation supports the additional predefined type FIXED in the package STANDARD. (See tests B86001C and B86001D.)

- Based literals.

  An implementation is allowed to reject a based literal with a value exceeding SYSTEM.MAX_INT during compilation, or it may raise NUMERIC_ERROR or CONSTRAINT_ERROR during execution. This implementation raises NUMERIC_ERROR during execution. (See test E24101A.)

- Expression evaluation.

  Apparently some default initialization expressions for record components are evaluated before any value is checked to belong to a component's subtype. (See test C32117A.)

  Assignments for subtypes are performed with the same precision as the base type. (See test C35712B.)

This implementation uses no extra bits for extra precision. This implementation uses all extra bits for extra range. (See test C35903A.)

Apparently NUMERIC_ERROR is raised when an integer literal operand in a comparison test is outside the range of the base type. (See test C45232A.)

Sometimes NUMERIC_ERROR is raised when an integer literal operand in a membership test is outside the range of the base type. (See test C45232A.)

Apparently NUMERIC_ERROR is raised when a literal operand in a fixed-point comparison or membership test is outside the range of the base type. (See test C45252A.)

Apparently underflow is not gradual. (See tests C45524A..Z.)

. Rounding.

The method used for rounding to integer is apparently round away from zero. (See tests C46012A..Z.)

The method used for rounding to longest integer is apparently round away from zero. (See tests C46012A..Z.)

The method used for rounding to integer in static universal real expressions is apparently round away from zero. (See test C4A014A.)

. Array types.

An implementation is allowed to raise NUMERIC_ERROR or CONSTRAINT_ERROR for an array having a 'LENGTH that exceeds STANDARD.INTEGER'LAST and/or SYSTEM.MAX_INT. For this implementation:

Declaration of an array type or subtype with more than SYSTEM.MAX_INT components raises NUMERIC_ERROR only for a two-dimensional array when the second dimension is the large number. Otherwise, no exception is raised. (See test C36003A.)

No exception is raised when 'LENGTH is applied to an array type with INTEGER'LAST + 2 components. (See test C36202A.)

No exception is raised when 'LENGTH is applied to an array type with SYSTEM.MAX_INT + 2 components. (See test C36202B.)

A packed BOOLEAN array having a 'LENGTH exceeding INTEGER'LAST raises no exception. (See test C52103X.)

A packed two-dimensional BOOLEAN array with more than INTEGER'LAST components raises CONSTRAINT_ERROR when the length of a dimension is calculated and exceeds INTEGER'LAST. (See test C52104Y.)

A null array with one dimension of length greater than INTEGER'LAST may raise NUMERIC_ERROR or CONSTRAINT_ERROR either when declared or assigned. Alternatively, an implementation may accept the declaration. However, lengths must match in array slice assignments. This implementation raises no exception. (See test E52103Y.)

In assigning one-dimensional array types, the expression appears to be evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. In assigning two-dimensional array types, the expression does not appear to be evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

. Discriminated types.

During compilation, an implementation is allowed to either accept or reject an incomplete type with discriminants that is used in an access type definition with a compatible discriminant constraint. This implementation accepts such subtype indications. (See test E38104A.)

In assigning record types with discriminants, the expression appears to be evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

. Aggregates.

In the evaluation of a multi-dimensional aggregate, index subtype checks appear to be made as choices are evaluated. (See tests C43207A and C43207B.)

In the evaluation of an aggregate containing subaggregates, not all choices are evaluated before being checked for identical bounds. (See test E43212B.)

All choices are evaluated before CONSTRAINT_ERROR is raised if a bound in a nonnull range of a nonnull aggregate does not belong to an index subtype. (See test E43211B.)

. Representation clauses.

An implementation might legitimately place restrictions on representation clauses used by some of the tests. If a representation clause is used by a test in a way that violates a restriction, then the implementation must reject it.

Enumeration representation clauses containing noncontiguous values for enumeration types other than character and boolean types are not supported. (See tests C35502I..J, C35502M..N, and A39005F.)

Enumeration representation clauses containing noncontiguous values for character types are not supported. (See tests C35507I..J, C35507M..N, and C55B16A.)

Enumeration representation clauses for boolean types containing representational values other than (FALSE => 0, TRUE => 1) are not supported. (See tests C35508I..J and C35508M..N.)

Length clauses with SIZE specifications for enumeration types are not supported provided that the size specified is at least 16 bits. (See test A39005B.)

Length clauses with STORAGE_SIZE specifications for access types are supported. (See tests A39005C and C87B62B.)

Length clauses with STORAGE_SIZE specifications for task types are supported. (See tests A39005D and C87B62D.)

Length clauses with SMALL specifications are supported. (See tests A39005E and C87B62C.)

Record representation clauses are supported provided that the components are aligned on 64-bit boundaries. (See test A39005G.)

Length clauses with SIZE specifications for derived integer types are supported. (See test C87B62A.)

. Pragmas.

The pragma INLINE is not supported for procedures or functions. (See tests LA3004A, LA3004B, EA3004C, EA3004D, CA3004E, and CA3004F.)

. Input/output.

The package SEQUENTIAL_IO cannot be instantiated with unconstrained array types and record types with discriminants without defaults. (See tests AE2101C, EE2201D, and EE2201E.)

The package DIRECT_IO cannot be instantiated with unconstrained array types and record types with discriminants without defaults. (See tests AE2101H, EE2401D, and EE2401G.)

Modes IN_FILE and OUT_FILE are supported for SEQUENTIAL_IO. (See tests CE2102D and CE2102E.)

Modes IN_FILE, OUT_FILE, and INOUT_FILE are supported for DIRECT_IO. (See tests CE2102F, CE2102I, and CE2102J.)

RESET and DELETE are supported for SEQUENTIAL_IO and DIRECT_IO. (See tests CE2102G and CE2102K.)

Dynamic creation and deletion of files are supported for SEQUENTIAL_IO and DIRECT_IO. (See tests CE2106A and CE2106B.)

Overwriting to a sequential file does not truncate the file. (See test CE2208B.)

An existing text file can be opened in OUT_FILE mode, can be created in OUT_FILE mode, and can be created in IN_FILE mode. (See test EE3102C.)

More than one internal file can be associated with each external file for text I/O for reading only. (See tests CE3111A..E (5 tests), CE3114B, and CE3115A.)

More than one internal file can be associated with each external file for sequential I/O for reading only. (See tests CE2107A..D (4 tests), CE2110B, and CE2111D.)

More than one internal file can be associated with each external file for direct I/O for reading only. (See tests CE2107F..I (5 tests), CE2110B, and CE2111H.)

An internal sequential access file and an internal direct access file cannot be associated with a single external file for writing. (See test CE2107E.)

An external file associated with more than one internal file cannot be deleted for SEQUENTIAL_IO, DIRECT_IO, and TEXT_IO. (See test CE2110B.)

Temporary sequential and direct files are given names. Temporary files given names are not deleted when they are closed. (See tests CE2108A and CE2108C.)

. Generics.

Generic subprogram declarations and bodies can be compiled in separate compilations provided that the body is compiled before any instantiations. (See tests CA1012A and CA2009F.)

Generic package declarations and bodies can be compiled in separate compilations provided that the body is compiled before any instantiations. (See tests CA2009C, BC3204C, and BC3205D.)

Generic unit bodies and their subunits can be compiled in separate compilations. (See test CA3011A.)

CHAPTER 3

TEST INFORMATION

3.1 TEST RESULTS

Version 1.9 of the ACVC comprises 3122 tests. When this compiler was tested, 27 tests had been withdrawn because of test errors. The AVF determined that 317 tests were inapplicable to this implementation. All inapplicable tests were processed during validation testing except for 229 executable tests that use floating-point precision exceeding that supported by the implementation. Modifications to the code, processing, or grading for ten tests were required to successfully demonstrate the test objective. (See section 3.6.)

The AVF concludes that the testing results demonstrate acceptable conformity to the Ada Standard.

3.2 SUMMARY OF TEST RESULTS BY CLASS

RESULT	TEST CLASS						TOTAL
	A	B	C	D	E	L	
Passed	105	1044	1556	17	12	44	2778
Inapplicable	5	7	297	0	6	2	317
Withdrawn	3	2	21	0	1	0	27
TOTAL	113	1053	1874	17	19	46	3122

3.3 SUMMARY OF TEST RESULTS BY CHAPTER

RESULT	CHAPTER 2	3	4	5	6	7	8	9	10	11	12	13	14	TOTAL
Passed	188	475	512	240	166	98	141	326	129	36	232	3	232	2778
Inapplicable	16	97	162	8	0	0	2	1	8	0	2	0	21	317
Withdrawn	2	14	3	0	0	1	2	0	0	0	2	1	2	27
TOTAL	206	586	677	248	166	99	145	327	137	36	236	4	255	3122

3.4 WITHDRAWN TESTS

The following 27 tests were withdrawn from ACVC Version 1.9 at the time of this validation:

B28003A	E28005C	C34004A	C35502P	A35902C
C35904A	C35904B	C35A03E	C35A03R	C37213H
C37213J	C37215C	C37215E	C37215G	C37215H
C38102C	C41402A	C45332A	C45614C	A74106C
C87B04B	C85018B	CC1311B	BC3105A	AD1A01A
CE2401H	CE3208A			

See Appendix D for the reason that each of these tests was withdrawn.

3.5 INAPPLICABLE TESTS

Some tests do not apply to all compilers because they make use of features that a compiler is not required by the Ada Standard to support. Others may depend on the result of another test that is either inapplicable or withdrawn. The applicability of a test to an implementation is considered each time a validation is attempted. A test that is inapplicable for one validation attempt is not necessarily inapplicable for a subsequent attempt. For this validation attempt, 317 tests were inapplicable for the reasons indicated:

- C35502I..J (2 tests), C35502M..N (2 tests), C35507I..J (2 tests), C35507M..N (2 tests), C35508I..J (2 tests), C35508M..N (2 tests), A39005F, and C55B16A use enumeration representation clauses which are not supported by this compiler.

- C35702A uses SHORT_FLOAT which is not supported by this implementation.

- A39005B uses length clauses with SIZE specifications for enumeration types which are not supported by this implementation.

- A39005G uses a record representation clause which is not supported by this implementation.

- The following 14 tests use SHORT_INTEGER, which is not supported by this implementation:

C45231B	C45304B	C45502B	C45503B	C45504B
C45504E	C45611B	C45613B	C45614B	C45631B
C45632B	B52004E	C55B07B	B55B09D	

- The following 13 tests use LONG_INTEGER, which is not supported by this implementation:

C45231C	C45304C	C45502C	C45503C	C45504C
C45504F	C45611C	C45613C	C45631C	C45632C
B52004D	C55B07A	B55B09C		

- C45231D and B86001D require a macro substitution for any predefined integer types other than INTEGER, SHORT_INTEGER, and LONG_INTEGER. This implementation does not support any such types.

- C45531M, C45531N, C45532M, and C45532N use fine 48-bit fixed-point base types which are not supported by this implementation.

- C45531O, C45531P, C45532O, and C45532P use coarse 48-bit fixed-point base types which are not supported by this implementation.

- C52008B declares a record type with four discriminants of type integer which have default values. The type may be used in the declaration of unconstrained objects, but the size of these objects exceeds the maximum object size of this implementation, and NUMERIC_ERROR is raised.

- C86001F redefines package SYSTEM, but TEXT_IO is made obsolete by this new definition in this implementation and the test cannot be executed since the package REPORT is dependent on the package TEXT_IO.

- C96001A assumes that SYSTEM.TICK <= DURATION'SMALL.

- CA2009C, CA2009F, BC3204C, and BC3205D contain instantiations of generics in cases where the body is not available at the time of the instantiation. This implementation creates a dependency on the missing body so that when the actual body is compiled, the unit containing the instantiation becomes obsolete.

- CA3004E, EA3004C, and LA3004A use the INLINE pragma for procedures, which is not supported by this implementation.

- CA3004F, EA3004D, and LA3004B use the INLINE pragma for functions, which is not supported by this implementation.

- AE2101C, EE2201D, and EE2201E use instantiations of package SEQUENTIAL_IO with unconstrained array types and record types having discriminants without defaults. These instantiations are rejected by this implementation.

- AE2101H, EE2401D, and EE2401G use instantiations of package DIRECT_IO with unconstrained array types and record types having discriminants without defaults. These instantiations are rejected by this implementation.

- CE2107B..E (4 tests), CE2107G..I (3 tests), CE2110B, CE2111D, CE2111H, CE3111B..E (4 tests), and CE3114B are inapplicable because multiple internal files cannot be associated with the same external file, when at least one is opened for writing. The proper exception is raised when multiple access is attempted.

  The following 229 tests require a floating-point accuracy that exceeds the maximum of 13 digits supported by this implementation:

C24113J..Y (16 tests)	C35705J..Y (16 tests)
C35706J..Y (16 tests)	C35707J..Y (16 tests)
C35708J..Y (16 tests)	C35802J..Z (17 tests)
C45241J..Y (16 tests)	C45321J..Y (16 tests)
C45421J..Y (16 tests)	C45521J..Z (17 tests)
C45524J..Z (17 tests)	C45621J..Z (17 tests)
C45641J..Y (16 tests)	C46012J..Z (17 tests)

3.6 TEST, PROCESSING, AND EVALUATION MODIFICATIONS

It is expected that some tests will require modifications of code, processing, or evaluation in order to compensate for legitimate implementation behavior. Modifications are made by the AVF in cases where legitimate implementation behavior prevents the successful completion of an (otherwise) applicable test. Examples of such modifications include: adding a length clause to alter the default size of a collection; splitting a Class B test into subtests so that all errors are detected; and confirming that messages produced by an executable test demonstrate conforming behavior that wasn't anticipated by the test (such as raising one exception instead of another).

Modifications were required for nine Class B tests, one Class C test, and one Class E test.

The following Class B tests were split because syntax errors at one point resulted in the compiler not detecting other errors in the test:

B27005A	BA3006A	BA3006B	BA3007B	BA3008A
BA3008B	BA3013A			

The following tests need a 'PRAGMA LIST (ON);' added at the beginning of the source file in order to have a complete source/error listing.

B28001R	B28001V	E28002D

C45651A requires that the result of the expression in line 227 be in the range given in line 228; however, this range excludes some acceptable results. This implementation passes all other checks of this test, and the AVO ruled the test is passed.

3.7 ADDITIONAL TESTING INFORMATION

3.7.1 Prevalidation

Prior to validation, a set of test results for ACVC Version 1.9 produced by the CRAY Ada Compiler was submitted to the AVF by the applicant for review. Analysis of these results demonstrated that the compiler successfully passed all applicable tests, and the compiler exhibited the expected behavior on all inapplicable tests.

3.7.2 Test Method

Testing of the CRAY Ada Compiler using ACVC Version 1.9 was conducted at Cray's facilities in Mendota Heights MN, by a validation team from the AVF. The configuration consisted of a CRAY-2 operating under UNICOS, Version 4.0.

A magnetic tape containing all tests except for withdrawn tests and tests requiring unsupported floating-point precisions was taken on-site by the validation team for processing. Tests that make use of implementation-specific values were customized before being written to the magnetic tape. Tests requiring modifications during the prevalidation testing were included in their modified form on the magnetic tape.

The contents of the magnetic tape were not loaded directly onto the host computer. The tape was loaded onto a Sun 3. All of the test files were then combined into a single file, using the UNIX tar facility. This single file was transferred through the local network to the host computer. Once on the host computer, the individual test files were extracted from the tar file. After the test files were loaded to disk, the full set of tests was compiled, linked and run on the CRAY-2. Results were combined into larger files and transferred through the local network to one of three machines for printing. The three machines included an AT and T 3B20, a Pyramid, and

an Amdahl.

The compiler was tested using command scripts provided by TELESOFT and reviewed by the validation team. The compiler was tested using all default option settings except for the following:

Class B tests

Option	Effect
-L	Generate source/error listing
-v	Output verbose progress messages

Executable tests

Option	Effect
-v	Output verbose progress messages
-m	Specify name of main program to link

Tests were compiled, linked, and executed (as appropriate) using a single host/target computer. Test output, compilation listings, and job logs were captured on magnetic tape and archived at the AVF. The listings examined on-site by the validation team were also archived.

3.7.3 Test Site

Testing was conducted at Mendota Heights MN and was completed on 12 June 1988.

APPENDIX A

DECLARATION OF CONFORMANCE

TELESOFT has submitted the following Declaration of Conformance concerning the CRAY Ada Compiler, Release 1.0.

DECLARATION OF CONFORMANCE

Compiler Implementor: TeleSoft, Inc.
Ada Validation Facility: ASD/SCEL, Wright-Patterson AFB, OH 45433-6503
Ada Compiler Validation Capability (ACVC), Version 1.9

Base Configuration

Base Compiler Name:	Cray Ada Compiler
Compiler Version.	1.0
Host Architecture ISA:	CRAY-2
OS & VER#:	UNICOS 4.0
Target Architecture ISA:	CRAY-2
OS & VER#:	UNICOS 4.0

Implementor's Declaration

I, the undersigned, representing TELESOFT, have implemented no deliberate extensions to the Ada Language Standard ANSI/MIL-STD-1815A in the compiler(s) listed in this declaration. I declare that Cray Research, Inc. is TeleSoft's licensee of the Ada language compiler(s) listed above and, as such, is responsible for maintaining said compiler(s) in conformance to ANSI/MIL-STD-1815A. All certificates and registrations for Ada language compiler(s) listed in this declaration shall be made only in the licensee's corporate name.

_____________________________ Date: June 21, 1988
TELESOFT
Raymond A. Parra, General Counsel
Director, Contracts

Licensee's Declaration

I, the undersigned, representing Cray Research, Inc. take full responsibility for implementation and maintenance of the Ada compiler(s) listed above, and agree to the public disclosure of the final Validation Summary Report. I further agree to continue to comply with the Ada trademark policy, as defined by the Ada Joint Program Office. I declare that all of the Ada language compiler(s) listed, and their host/target performance are in compliance with the Ada Language Standard ANSI/MIL-STD-1815A

_____________________________ Date: June 13, 1988
Cray Research, Inc.
Bruce White
Ada Project Manager

APPENDIX B

APPENDIX F OF THE Ada STANDARD

The only allowed implementation dependencies correspond to implementation-dependent pragmas, to certain machine-dependent conventions as mentioned in chapter 13 of the Ada Standard, and to certain allowed restrictions on representation clauses. The implementation-dependent characteristics of the CRAY Ada Compiler, Release 1.0, are described in the following sections, which discuss topics in Appendix F of the Ada Standard. Implementation-specific portions of the package STANDARD are also included in this appendix.

```
package STANDARD is

     ...

     type INTEGER is range -(2**45) .. (2**45)-1;

     type FLOAT is digits 13 range 6.52530E-55 .. 1.53249E+54;

     type FIXED is delta 2#1.0#E-45 range -1.0 .. 1.0-2#1.0#E-45;

     type DURATION is delta 2#1.0#E-14 range -86400.0 .. 86400.0;

     ...

end STANDARD;
```

APPENDIX F

1. Predefined Pragma

pragma LIST(ON OFF);
It may appear anywhere a pragma is allowed. The pragma has the effect of generating the source compilation. The listing will begin at the first pragma list(ON) statement if no previous pragma list(OFF) statement was encountered. Otherwise, the listing will begin at the top of the source.

Implementation Dependent Pragmas

pragma COMMENT(<string_literal>);
It may only appear within a compilation unit. The pragma comment has the effect of embedding the given sequence of characters in the object code of the compilation unit.

pragma LINKNAME(<subprogram_name>, <string_literal>);
It may appear in any declaration section of a unit. This pragma must also appear directly after an interface pragma for the same <subprogram_name>. The pragma linkname has the effect of making string_literal apparent to the linker.

2. Implementation Dependent Attributes

There are no implementation dependent attributes.

3. Specification of Package SYSTEM

```
PACKAGE System IS

 TYPE Address is PRIVATE;
 TYPE Subprogram_Value is PRIVATE;

 TYPE Name    IS (CRAY_XMP, CRAY_2);

 System_Name  : CONSTANT name := CRAY_2;

 Storage_Unit : CONSTANT := 64;
 Memory_Size  : CONSTANT := (2 ** 24) - 1;
```

APPENDIX F, Cont.

```
-- System-Dependent Named Numbers:

Min_Int      : CONSTANT := -(2 ** 45);
Max_Int      : CONSTANT := (2 ** 45) - 1;
Max_Digits   : CONSTANT := 13;
Max_Mantissa : CONSTANT := 45;
Fine_Delta   : CONSTANT := 1.0   (2 ** Max_Mantissa);
Tick         : CONSTANT := 10.0E-3;

-- Other System-Dependent Declarations

SUBTYPE Priority IS Integer RANGE 0 .. 63;

Max_Text_Io_Count : CONSTANT := Max_Int-1;
Max_Text_Io_Field : CONSTANT := 1000;

PRIVATE

 TYPE Address is Access Integer;

 TYPE Subprogram_Value IS
  RECORD
   Proc_addr    : Address;
   Static_link  : Address;
   Global_frame : Address;
  END RECORD;

END System;
```

4. Restrictions on Representation Clauses

The Compiler supports the following representation clauses:

Length Clauses: for enumeration and derived integer types 'SIZE attribute (LRM 13.2(a))
Length clauses: for access types 'STORAGE_SIZE attribute (LRM13.2(b))
Length Clauses: for tasks types 'STORAGE_SIZE attribute (LRM 13.2(c))
Length clauses: for fixed point types 'SMALL attribute (LRM13.2(d))
Enumeration clauses: for character and enumeration types other than character and boolean (LRM 13.3)
Record representation clauses (LRM 13.4)
Address Clauses: for objects and entries (LRM 13.5(a)(c))

This compiler does NOT support the following representation clauses:

Enumeration clauses: for boolean (LRM 13.3)

APPENDIX F, Cont.

Address clauses for subprograms, packages, and tasks (LRM 13.5(b))

Note: The Cray CRAY-2 compiler contains a restriction that allocated objects must have a minimum allocation size of 64 bits.

5. Implementation dependent naming conventions

There are no implementation-generated names denoting implementation dependent components.

6. Expressions that appear in address specifications are interpreted as the first storage unit of the object.

7. Restrictions on Unchecked Conversions

Unchecked conversions are allowed between any types unless the target type is an unconstrained record or array type.

8. I/O Package Characteristics

Instantiations of DIRECT_IO and SEQUENTIAL_IO are supported with the following exceptions:

* Unconstrained array types.

* Unconstraintеd types with discriminants without default values.

* Multiple internal files opened to the same external file may only be opened for reading.

* In DIRECT_IO the type COUNT is defined as follow:

  type COUNT is range 0..2**45-1;

* In TEXT_IO the type COUNT is defined as follows:

  type COUNT is range 0..2**45-3;

* In TEXT_IO the subtype FIELD is defined as follows:

  subtype FIELD is INTEGER range 0..1000;

9. Definition of STANDARD

STANDARD is not an Ada package with a specification in our implementation. Our compilation system does not compile any source corresponding to the predefined package STANDARD. In fact, STANDARD cannot generally be written fully using standard Ada because the definitions of predefined numeric types like INTEGER and FLOAT require specification of properties that cannot be defined by means of Ada type declarations. It would probably be more appropriate for the AVO to request implementations to provide the names of all predefined numeric types and the values of their various attributes instead of asking for some artificially constructed source for STANDARD, especially since the predefined numeric types are the only declarations of allowed variation within the package. The generation of package STANDARD in our implementation is achieved by means of a special text file that specifies the names and certain attribute values for the various numeric types supported by the target configuration.

For this target system the numeric types and their properties are as follows:

Integer types:

INTEGER

size = 64
first = -35,184,372,088,832 = -2**45
last = +35,184,372,088,831 = 2**45-1

Floating-point types:

FLOAT

size = 64
digits = 13
'small = 6.52530E-55
'large = 1.53249E+54
machine_radix = 2
machine_mantissa = 45
machine_emin = -16384
machine_emax = +16383

Fixed-point types:

FIXED

size = 64
delta = 2#1.0#e-45
first = -1.00000
last = +1.0 - 2#1.0#e-45

DURATION

size = 64
delta = 2#1.0#e-14
first = -86400
last = +86400

APPENDIX C

TEST PARAMETERS

Certain tests in the ACVC make use of implementation-dependent values, such as the maximum length of an input line and invalid file names. A test that makes use of such values is identified by the extension .TST in its file name. Actual values to be substituted are represented by names that begin with a dollar sign. A value must be substituted for each of these names before the test is run. The values used for this validation are given below.

Name and Meaning	Value
$BIG_ID1 Identifier the size of the maximum input line length with varying last character.	(1..199 => 'A', 200 => '1')
$BIG_ID2 Identifier the size of the maximum input line length with varying last character.	(1..199 => 'A', 200 => '2')
$BIG_ID3 Identifier the size of the maximum input line length with varying middle character.	(1..100 | 102..200 => 'A', 101 => '3')
$BIG_ID4 Identifier the size of the maximum input line length with varying middle character.	(1..100 | 102..200 => 'A', 101 => '4')
$BIG_INT_LIT An integer literal of value 298 with enough leading zeroes so that it is the size of the maximum line length.	(1..197 => '0', 198..200 => "298")

Name and Meaning	Value
$BIG_REAL_LIT A universal real literal of value 690.0 with enough leading zeroes to be the size of the maximum line length.	(1..194 => '0', 195..200 => "69.0E1")
$BIG_STRING1 A string literal which when catenated with BIG_STRING2 yields the image of BIG_ID1.	(1 => '"', 2..101 => 'A', 102 => '"')
$BIG_STRING2 A string literal which when catenated to the end of BIG_STRING1 yields the image of BIG_ID1.	(1 => '"', 2..100 => 'A', 101..102 => "1""")
$BLANKS A sequence of blanks twenty characters less than the size of the maximum line length.	(1..180 => ' ')
$COUNT_LAST A universal integer literal whose value is TEXT_IO.COUNT'LAST.	(2**45)-3
$FIELD_LAST A universal integer literal whose value is TEXT_IO.FIELD'LAST.	1000
$FILE_NAME_WITH_BAD_CHARS An external file name that either contains invalid characters or is too long.	BAD-CHARSX}]!@#$^&~Y
$FILE_NAME_WITH_WILD_CARD_CHAR An external file name that either contains a wild card character or is too long.	WILD-CHAR*.NAM
$GREATER_THAN_DURATION A universal real literal that lies between DURATION'BASE'LAST and DURATION'LAST or any value in the range of DURATION.	100_000.0

Name and Meaning	Value
$GREATER_THAN_DURATION_BASE_LAST A universal real literal that is greater than DURATION'BASE'LAST.	131_073.0
$ILLEGAL_EXTERNAL_FILE_NAME1 An external file name which contains invalid characters.	BADCHAR*^/%
$ILLEGAL_EXTERNAL_FILE_NAME2 An external file name which is too long.	(1..266 => 'A')
$INTEGER_FIRST A universal integer literal whose value is INTEGER'FIRST.	-(2**45)
$INTEGER_LAST A universal integer literal whose value is INTEGER'LAST.	(2**45)-1
$INTEGER_LAST_PLUS_1 A universal integer literal whose value is INTEGER'LAST + 1.	(2**45)
$LESS_THAN_DURATION A universal real literal that lies between DURATION'BASE'FIRST and DURATION'FIRST or any value in the range of DURATION.	-100_000.0
$LESS_THAN_DURATION_BASE_FIRST A universal real literal that is less than DURATION'BASE'FIRST.	-131_073.0
$MAX_DIGITS Maximum digits supported for floating-point types.	13
$MAX_IN_LEN Maximum input line length permitted by the implementation.	200
$MAX_INT A universal integer literal whose value is SYSTEM.MAX_INT.	(2**45)-1
$MAX_INT_PLUS_1 A universal integer literal whose value is SYSTEM.MAX_INT+1.	2**45

Name and Meaning	Value
$MAX_LEN_INT_BASED_LITERAL A universal integer based literal whose value is 2#11# with enough leading zeroes in the mantissa to be MAX_IN_LEN long.	(1..2 => "2:", 3..197 => '0', 198..200 => "11:")
$MAX_LEN_REAL_BASED_LITERAL A universal real based literal whose value is 16:F.E: with enough leading zeroes in the mantissa to be MAX_IN_LEN long.	(1..3 => "16:", 4..196 => '0', 197..200 => "F.E:")
$MAX_STRING_LITERAL A string literal of size MAX_IN_LEN, including the quote characters.	(1 => '"', 2..199 => 'A', 200 => '"')
$MIN_INT A universal integer literal whose value is SYSTEM.MIN_INT.	-(2**45)
$NAME A name of a predefined numeric type other than FLOAT, INTEGER, SHORT_FLOAT, SHORT_INTEGER, LONG_FLOAT, or LONG_INTEGER.	NO_SUCH_TYPE
$NEG_BASED_INT A based integer literal whose highest order nonzero bit falls in the sign bit position of the representation for SYSTEM.MAX_INT.	16#FFFFFFFE#

APPENDIX D

WITHDRAWN TESTS

Some tests are withdrawn from the ACVC because they do not conform to the Ada Standard. The following 27 tests had been withdrawn at the time of validation testing for the reasons indicated. A reference of the form "AI-ddddd" is to an Ada Commentary.

- B28003A: A basic declaration (line 36) incorrectly follows a later declaration.

- E28005C: This test requires that "PRAGMA LIST (ON);" not appear in a listing that has been suspended by a previous "PRAGMA LIST (OFF);"; the Ada Standard is not clear on this point, and the matter will be reviewed by the AJPO.

- C34004A: The expression in line 168 yields a value outside the range of the target type T, but there is no handler for CONSTRAINT_ERROR.

- C35502P: The equality operators in lines 62 and 69 should be inequality operators.

- A35902C: The assignment in line 17 of the nominal upper bound of a fixed-point type to an object raises CONSTRAINT_ERROR, for that value lies outside of the actual range of the type.

- C35904A: The elaboration of the fixed-point subtype on line 28 wrongly raises CONSTRAINT_ERROR, because its upper bound exceeds that of the type.

- C35904B: The subtype declaration that is expected to raise CONSTRAINT_ERROR when its compatibility is checked against that of various types passed as actual generic parameters, may, in fact, raise NUMERIC_ERROR or CONSTRAINT_ERROR for reasons not anticipated by the test.

- C35A03E and C35A03R: These tests assume that attribute 'MANTISSA returns 0 when applied to a fixed-point type with a null range, but the Ada Standard does not support this assumption.

- C37213H: The subtype declaration of SCONS in line 100 is incorrectly expected to raise an exception when elaborated.

- C37213J: The aggregate in line 451 incorrectly raises CONSTRAINT_ERROR.

- C37215C, C37215E, C37215G, and C37215H: Various discriminant constraints are incorrectly expected to be incompatible with type CONS.

- C38102C: The fixed-point conversion on line 23 wrongly raises CONSTRAINT_ERROR.

- C41402A: The attribute 'STORAGE_SIZE is incorrectly applied to an object of an access type.

- C45332A: The test expects that either an expression in line 52 will raise an exception or else MACHINE_OVERFLOWS is FALSE. However, an implementation may evaluate the expression correctly using a type with a wider range than the base type of the operands, and MACHINE_OVERFLOWS may still be TRUE.

- C45614C: The function call of IDENT_INT in line 15 uses an argument of the wrong type.

- A74106C, C85018B, C87B04B, and CC1311B: A bound specified in a fixed-point subtype declaration lies outside of that calculated for the base type, raising CONSTRAINT_ERROR. Errors of this sort occur at lines 37 & 59, 142 & 143, 16 & 48, and 252 & 253 of the four tests, respectively.

- BC3105A: Lines 159 through 168 expect error messages, but these lines are correct Ada.

- AD1A01A: The declaration of subtype SINT3 raises CONSTRAINT_ERROR for implementations which select INT'SIZE to be 16 or greater.

- CE2401H: The record aggregates in lines 105 and 117 contain the wrong values.

- CE3208A: This test expects that an attempt to open the default output file (after it was closed) with mode IN_FILE raises NAME_ERROR or USE_ERROR; by Commentary AI-00048, MODE_ERROR should be raised.